Chapter 1

Introduction

> *"Jealous stepmother and sisters; magical aid by a beast; a marriage won by gifts magically provided; a bird revealing a secret; a recognition by aid of a ring; or show; or what not; a démouement of punishment; a happy marriage - all those things, which in sequence, make up Cinderella, may and do occur in an incalculable number of other combinations. "*
>
> — MR. Cox ***1893**, Cinderella: Three hundred and forty-five variants [87]*

The W3C standarized bytecode for the web environment with WebAssembly (Wasm) language in 2015, opening up the browser clients ecosystem to a broader collection of programming languages. Wasm allows the use of existing programs or libraries that are written in other languages, such as C and Rust, to be run in the web browser environment. It also claims that is better than JavaScript to perform compute-intensive tasks [41]. Further browser environments, it evolves to be a new technology for Edge-Cloud computing platforms, resulting in bandwidth saving, execution improvement, and process-on-demand fast spawning [4, 14]. However, since it a relatively new technology, it is not exempt of vulnerabilities.

Software bugs are inherited in any software development process, including both, the WebAssembly engines and the source code that generates the Wasm binaries. One temporary solution to deal with bugs is to move them in time as a preemptive solution, the bug is only available in a time window. This strategy is usually called Moving Target Defense (MTD). Moving Target Defense for software was first proposed as a collection of techniques that aim to improve the security of a system by constantly moving its vulnerable components [12, 54]. Usually, MTD techniques revolve around changing systems to reduce vulnerable surfaces. This increases uncertainty for attackers and makes their attacks more difficult. Ultimately, potential attackers cannot hit what they cannot see. MTD can be implemented in different ways, including via Software Diversification.

Software Diversification has shown to be a good preemptive technique [?] , preventing exploitation of vulnerabilities or hardening harmful analysis of programs. In one of our experiments, a CVE was discovered inside the Lucet compiler, showing new vulnerabilities are still present in battle-tested engines. This CVE

was discovered thanks to our Software Diversification strategies and was solved even before it was public.

The low presence of defenses implementations for WebAssembly and the pace of their practical adoption motivate our work on Software Diversification as a preemptive technique that can help against known and yet unknown vulnerabilities. Another motivation for this work is that WebAssembly lacks of natural diversity, *i.e.,* for a single WebAssembly program in the wild, it has a lower presence of versions. Moreover, compared to the work of Harrand et al. [3], in WebAssembly, we cannot use preexisting and different programs to provide diversification. In fact, according to the work of Hilbig et al. [8], they filter out half of internet available WebAssembly corpus due to that filtered programs where artifically provided by one of our tools.

1.1 Research questions

In this section, we present our three research questions. Our research questions are formulated by merging our publications and experiences during the creation of Software Diversification for WebAssembly.

RQ_1 **To what extent can we artifically generate program variants for WebAssembly?**
With this research question, we quantitatively assess the static differences between program variants created by our approach. We answer this question at the population level, where a program population is the collection of one original program and its generated variants. We aim to investigate the code properties that increases(or diminishes) generated diversification at population level.

RQ_2 **To what extent are the generated variants dynamically different?**
With this research question, we complement RQ_1. We aim to investigate the impact on execution traces and execution times of the generated program variants.

RQ_3 **To what extent do the artificial variants exhibit different execution times on Edge-Cloud platforms?**
With this research question, we aim to investigate the impact of Software Diversification for WebAssembly in an emerging technology, Edge-Cloud computing. We evaluate the impact of a novel multivariant execution approach on real-world WebAssembly programs in a world-wide scale experiment.

1.2 Contributions

The contributions of this thesis are:

C_1 Technical contribution: Along with this work, we contribute with several software artifacts and summarize the main challenges faced during their implementation.

C_2 Methodological contribution: We propose a quantitative methodology to evaluate the impact of our artifacts, assessing the creation of Artificial Software Diversification for WebAssembly.

C_3 Experimental contribution: We contribute to generate Artificial Software Diversification for WebAssembly. We empirically demonstrate the impact on static and dynamic behavior for our diversification technique.

C_4 Theoretical contribution: We summarize the code transformations used to artifically generate software diversification through an exhaustive literature review and highlight the lack of diversification techniques for WebAssembly. In addition, we discuss the incorporation of *Constant Inferring* as a new diversification technique.

1.3 Publications

This work is based on the following publications:

P_1 Superoptimization of WebAssembly Bytecode [23]
Javier Cabrera-Arteaga, Shrinish Donde, Jian Gu, Orestis Floros, Lucas Satabin, Benoit Baudry, Martin Monperrus
Programming 2020, MoreVMs'20
Abstract: Motivated by the fast adoption of WebAssembly, we propose the first functional pipeline to support the superoptimization of WebAssembly bytecode. Our pipeline works over LLVM and Souper. We evaluate our superoptimization pipeline with 12 programs from the Rosetta code project. Our pipeline improves the code section size of 8 out of 12 programs. We discuss the challenges faced in superoptimization of WebAssembly with two case studies.

P_2 CROW: Code Diversification for WebAssembly [10]
Javier Cabrera-Arteaga, Orestis Floros, Oscar Vera-Pérez, Benoit Baudry, Martin Monperrus
NDSS 2021, MADWeb
Abstract: The adoption of WebAssembly has rapidly increased in the last few years as it provides a fast and safe model for program execution. However, WebAssembly is not exempt from vulnerabilities that could be exploited by side channels attacks. This class of vulnerabilities that can be addressed by code diversification. In this paper, we present the first fully automated workflow for the diversification of WebAssembly binaries. We present CROW, an open-source tool implementing this workflow. We evaluate CROW's capabilities on 303 C programs and study its use on a real-life security-sensitive program: libsodium, a cryptographic library.

Overall, CROW is able to generate diverse variants for 239 out of 303,(79%) small programs. Furthermore, our experiments show that our approach and tool is able to successfully diversify off-the-shelf cryptographic software (libsodium).

P_3 Multi-Variant Execution at the Edge [9]
Javier Cabrera-Arteaga, Pierre Laperdrix, Martin Monperrus, Benoit Baudry
Under review
Abstract: Edge-cloud computing offloads parts of the computations that traditionally occurs in the cloud to edge nodes,e.g., CDN servers, in order to get closer to the users and reduce latency. To improve performance even further, WebAssembly is increasingly used in this context. Edge-cloud computing providers, such as Fastly or Cloudflare, let their clients deploy stateless services in the form of WebAssembly binaries, which are then translated to machine code and sandboxed for a safe execution at the edge. In this context, we propose a technique that (i) automatically diversifies WebAssembly binaries that are deployed to the edge and (ii) randomizes execution paths at runtime, turning the execution of the services into a moving target. Given a service tobe deployed at the edge, we automatically synthesize functionally equivalent variants for the functions that implement the service.All the variants are then wrapped into a single multivariant WebAssembly binary. When the service endpoint is executed,every time a function is invoked, one of its variants is randomly selected. We implement this technique in the MEWE tool and we validate it with 7 services for cryptography and QR encoding. MEWE generates multivariant binaries that embed hundreds of function variants. We execute the multivariant binaries on the worldwide edge platform provided by Fastly. We show that,at runtime, the multivariant exhibit a remarkable diversity ofexecution traces, across the whole edge platform.

Other publications and talks

1. Scalable Comparison of JavaScript V8 Bytecode Traces [27]
Javier Cabrera-Arteaga, Martin Monperrus, Benoit Baudry
SPLASH 2019, VMIL
Abstract: The comparison and alignment of runtime traces are essential, e.g., for semantic analysis or debugging. However, naive sequence alignment algorithms cannot address the needs of the modern web: (i) the bytecode generation process of V8 is not deterministic; (ii) bytecode traces are large. We present STRAC, a scalable and extensible tool tailored to compare bytecode traces generated by the V8 JavaScript engine. Given two V8 bytecode traces and a distance function between trace events, STRAC computes and provides the best alignment. The key insight is to split access between memory and disk. STRAC can identify semantically equivalent web pages and is capable of processing huge V8 bytecode traces whose order of magnitude matches today's web like https://2019.splashcon.org, which generates approx. 150k of V8 bytecode instructions.

2. (Talk) Wasm-mutate: Fuzzing WebAssembly Compilers with E-Graphs
   Javier Cabrera-Arteaga, Nicholas Fitzgerald, Martin Monperrus, Benoit Baudry
   PLDI 2022, EGRAPHs

Thesis layout

This dissertation is organized in five chapters including this. Chapter 2 presents background and the state of the art for WebAssembly and Artificial Software Diversification. Chapter 3 describes our technical contributions, faced challenges and engineering decisions carried out to implement our artifacts. Chapter 4 describes the methodology followed to answer the three main research questions driving this thesis. Chapter 5 details the main results of this work. Chapter 6 concludes and discuss future work. In addition, this dissertation contains the collection of research papers previously mentioned in this chapter.